THE UNIVERSITY OF NEW MEXICO

ECE 538 Advanced Computer Architecture

Instructor: Lei Yang

Department of Electrical and Computer Engineering

November 9, 2021

Exception and Interrupt

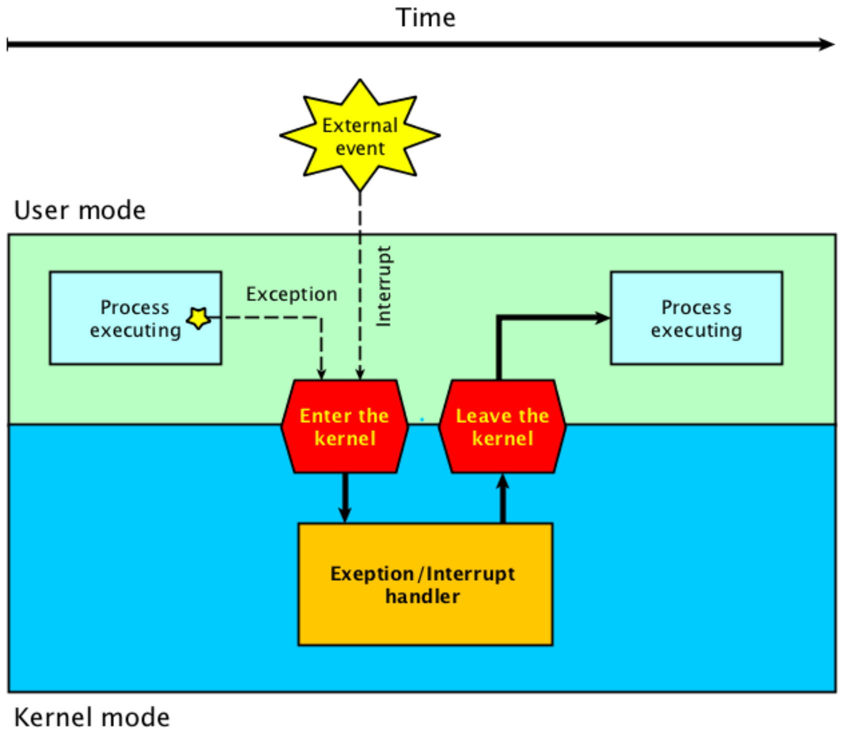
Referred to Chapter appendix A.7 (Page A-33)

in Book “Computer Organization and Design: The Hardware/ Software Interface, Fifth Edition”

EXCEPTION AND INTERRUPT

- Trap
  - Software instruction that causes execution of exception handlers and interrupt service routines
- Exception
  - Arises within the CPU

    -- e.g., undefined opcode, overflow, …
- Interrupt
  - From an external I/O controller

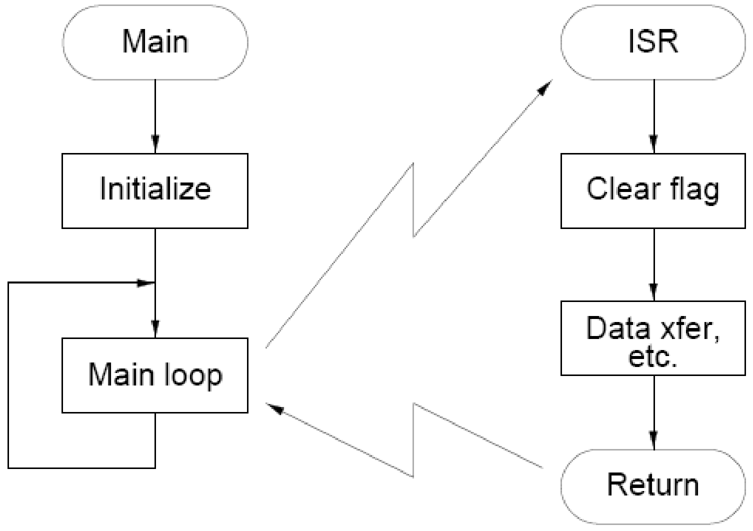


Exception and Interrupt

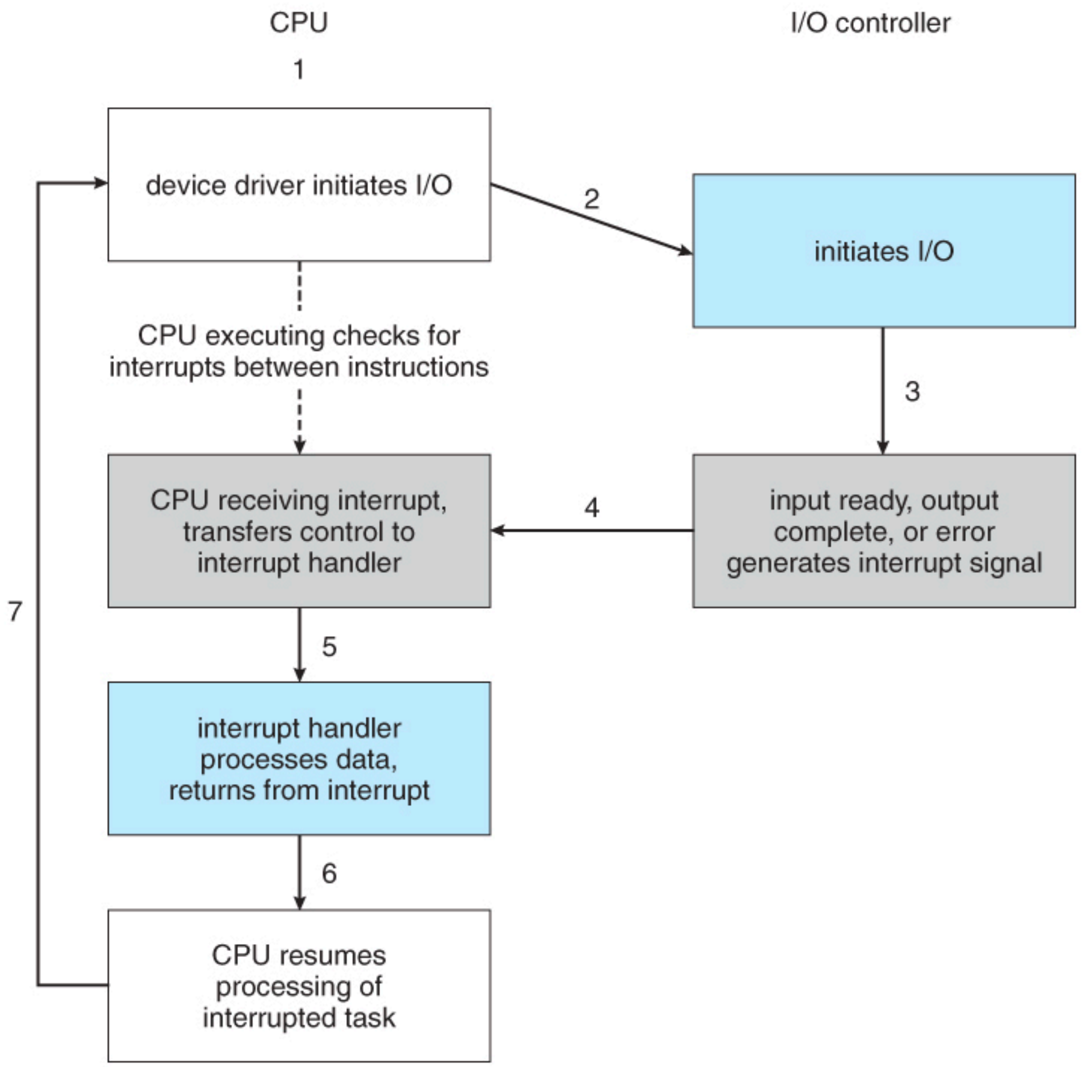
- Interrupt: are signals sent to the CPU by external devices, normally I/O devices. They tell the CPU to stop its current activities and execute the appropriate part of the operating system
  - Hardware Interrupts: generated by hardware devices to signal that they need some attention from the OS.
  - Software Interrupts: generated by programs when they want to request a system call to be performed by the operating system.
  - Traps: generated by the CPU itself to indicate that some error or condition occurred for which assistance from the operating system is needed.

Exception and Interrupt

- General idea of interrupt



- A main program runs continuously and is completely oblivious to the needs of any IO device.
- An *interrupt service routine* (ISR) is called when an interrupt occurs (for example, an IO device is ready).


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
input ready, output complete, or error generates interrupt signal
4
CPU receiving interrupt, transfers control to interrupt handler
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task
7

EXCEPTION AND INTERRUPT

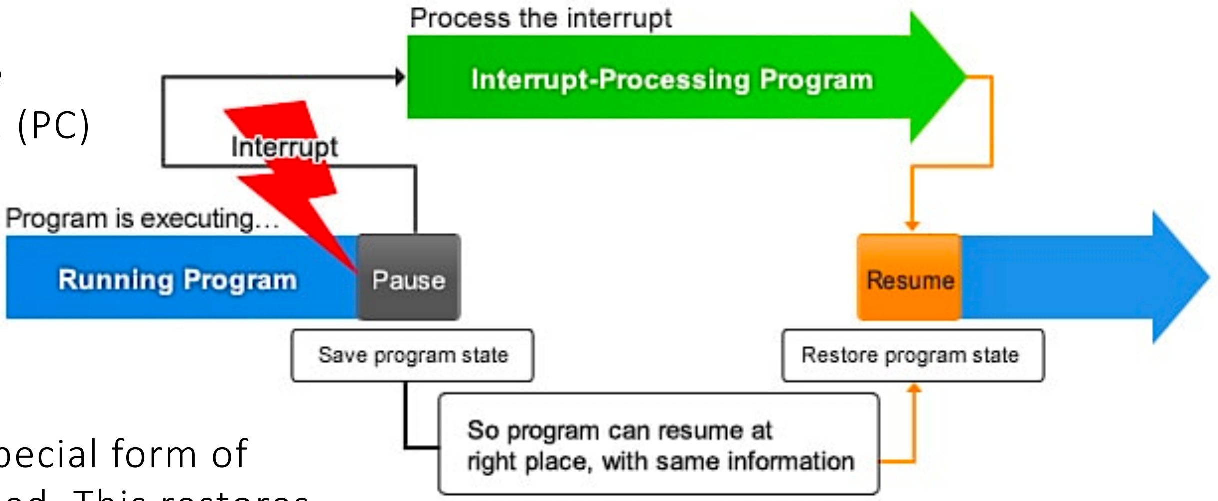
- General idea of interrupt
  - Hardware devices signal operating system when events occur
  - Operating system *preempts* any running process (switch context away from the running process) to handle the event
  - Allow for a more responsive system than would be available with polling
  - More complex implementation (require hardware support)

- Interrupt actions:

Step 1: All processor statuses are saved automatically on the stack. (PC)

Step 2: Control is transferred to a special address specified by an Interrupt Vector (a double byte that gives address of the ISR)

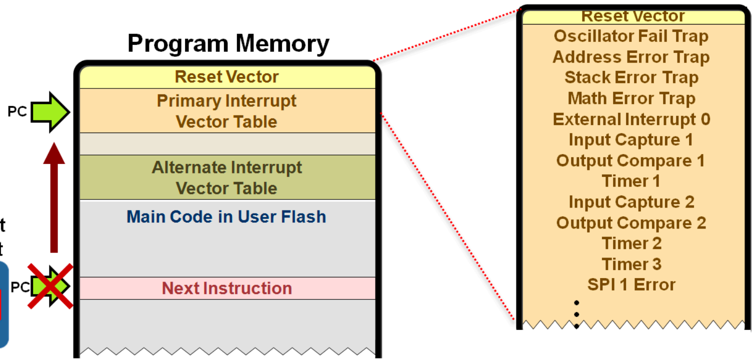
Step 3: At the end of the ISR, a special form of return instruction RTI must be used. This restores the processor status

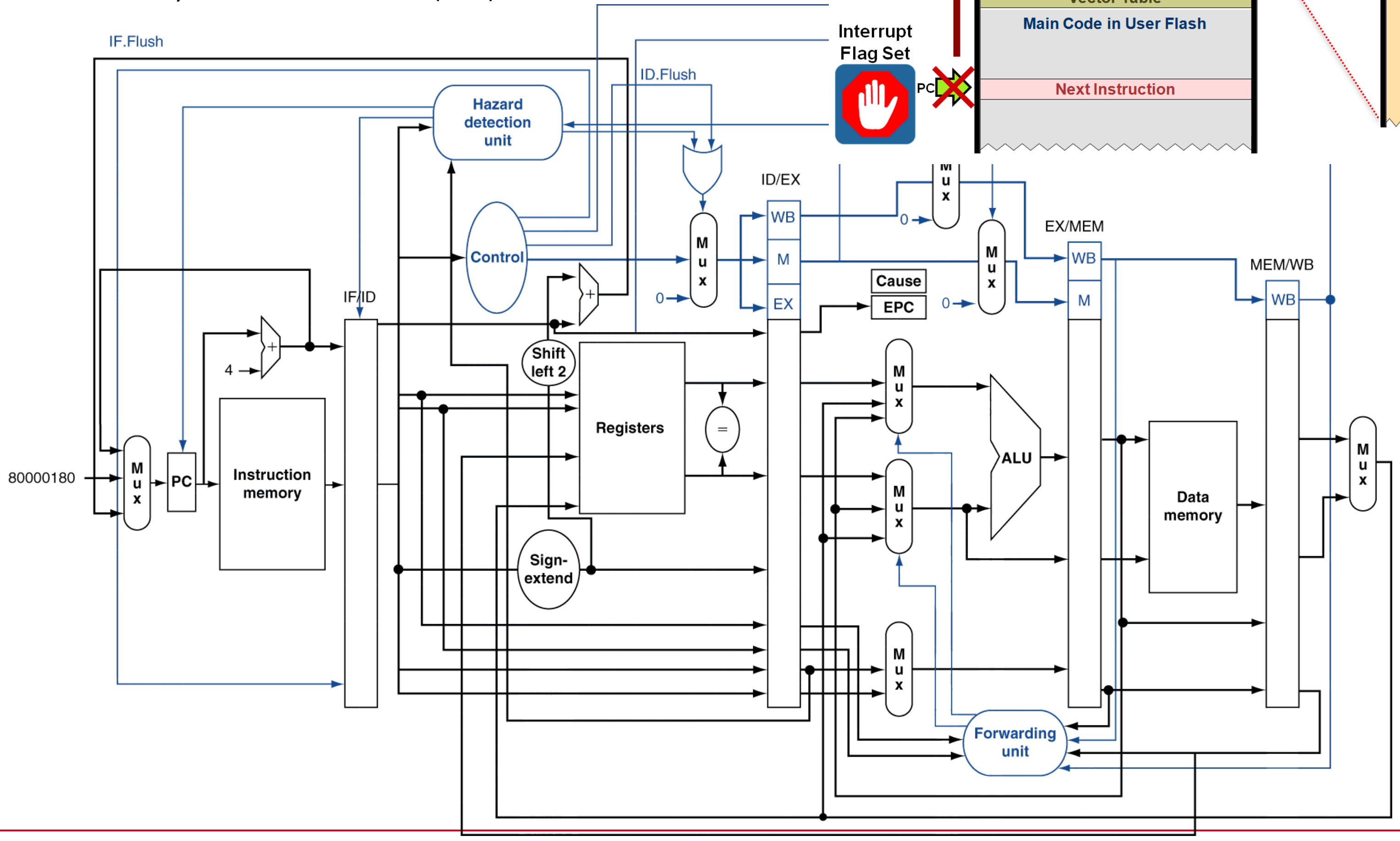


Exception and Interrupt

- Interrupt actions:

Step 1: All processor statuses are saved automatically on the stack. (PC)





EXCEPTION AND INTERRUPT

- Interrupt actions:

Step 2: Control is transferred to a special address specified by an Interrupt Vector (a double byte that gives address of the ISR)

Vector Address	Interrupt Source, Flag
$FFFE	Reset
$FFFC	COP Clock Monitor Fail Reset
$FFFA	COP Failure Reset
$FFF8	Unimplemented Instruction Trap
$FFF6	SWI
$FFF4	XIRQ
$FFF2	IRQ or Parallel IO handshake, STAF
$FFF0	Real-time Interrupt, RTIF
$FFEE	Timer input capture 1, IC1F
$FFEC	Timer input capture 2, IC2F
$FFEA	Timer input capture 3, IC3F
$FFE8	Timer output compare 1 OC1F

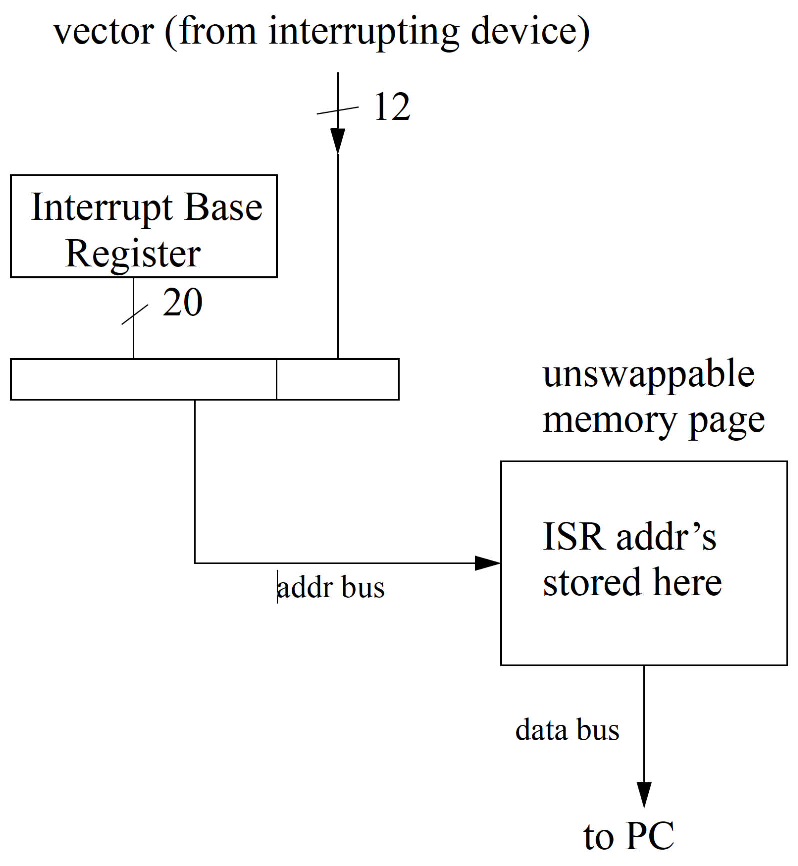
EXCEPTION AND INTERRUPT

- Interrupt actions:

Step 3: At the end of the ISR, a special form of return instruction RTI (Return Interrupt) must be used. This restores the processor status.

In MIPS, the instruction is ***Returned From Exception (RFE)***

=== Initialize Interrupt Vector===



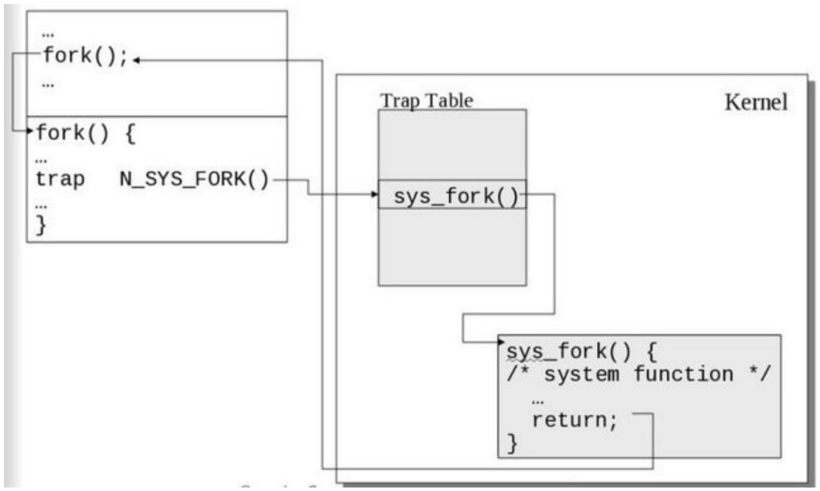
Interrupt Vector Implementation

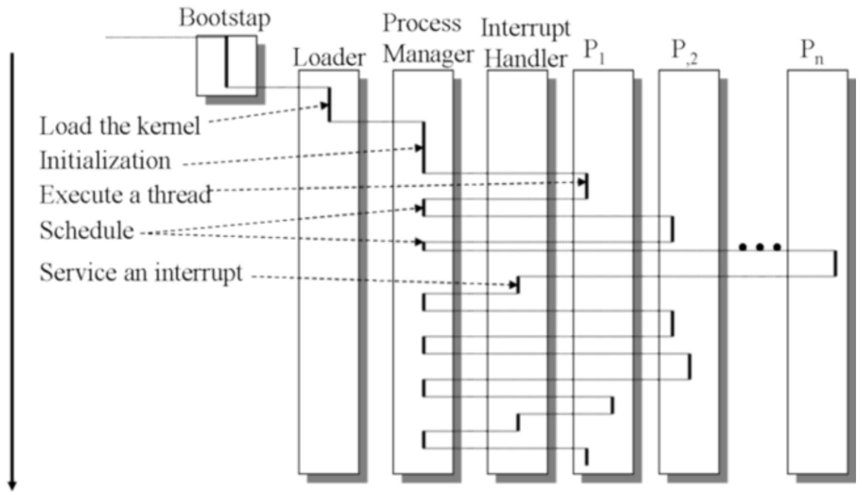
Exception and Interrupt

- CPU Response to Interrupts
  - Using the pointer to the current process control block, the state and all register values for the process are saved for use when the process is later restarted.
  - The CPU mode bit is switched to supervisory mode.
  - Using the pointer to the interrupt handler table and the interrupt vector, the location of the kernel code to execute is determined. The interrupt vector is the IRQ for hardware interrupts (read from an interrupt controller register) and an argument to the interrupt assembly language instruction for software interrupts.
  - Processing is switched to the appropriate portion of the kernel.

EXCEPTION AND INTERRUPT

- CPU Response to Interrupts
  - The CPU uses a table and the interrupt vector to find OS the code to execute in response to interrupts.





- As the computer runs, processing switches between user processes and the operating system as hardware and software interrupts are received.

EXCEPTION AND INTERRUPT

- Types of Exception
  - I/O device request
  - Invoking an operating system service from a user program (system call)
  - Tracing instruction execution
  - Breakpoint (programmer-requested interrupt)
  - Integer arithmetic overflow or underflow; FP arithmetic anomaly
  - Page fault
  - Misaligned memory accesses (if alignment is required)
  - Memory protection violation
  - Using an undefined instruction
  - Hardware malfunction
  - Power failure
  - …

Exception and Interrupt

- Requirements on Exception
  - Synchronous versus asynchronous
  - User requested versus coerced
  - User maskable versus user nonmaskable
  - Within versus between instructions
  - Resume versus terminate

EXCEPTION AND INTERRUPT

- Types of Exception in MIPS:
  - Arithmetic overflow
  - Undefined instruction
  - System call

Exception and Interrupt

Handling Exception in MIPS:

- In MIPS, exceptions managed by a System Control Coprocessor (CP0)
- Save PC of offending (or interrupted) instruction
  - In MIPS: Exception Program Counter (EPC)
- Save indication of the problem
  - In MIPS: Cause register
  - We'll assume 1-bit
    - 0 for undefined opcode, 1 for overflow

Exceptions in MIPS

Pipeline Stage	Problem exceptions occurring
IF	Page fault on instruction fetch; misaligned memory access; memory protection violation
ID	Undefined or illegal opcode
EX	Arithmetic exception
MEM	Page fault on data fetch; misaligned memory access; memory protection violation
WB	None

- Multiple executions might occur since multiple instructions are executing (LW followed by DIV might cause page fault in the same cycle)
- Exceptions can even occur out of order (e.g., a page fault or instruction memory can occur earlier than a page fault of data memory caused by the proceeding instruction in the pipeline)

Pipeline execution has to be handled in order of execution of faulting instructions not according to the time they occur

EXCEPTION AND INTERRUPT

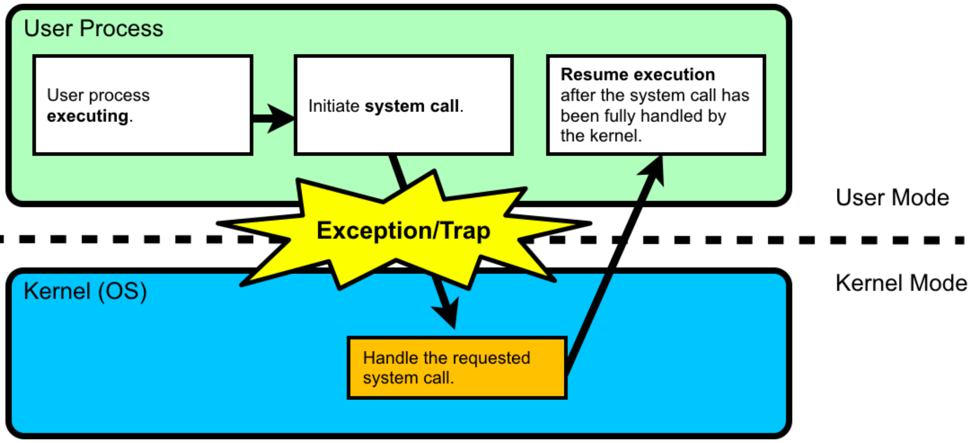
- Handling Exception in MIPS:
  - Step 1: Read cause, and transfer to relevant handler
  - Step 2: Determine action required

    If restartable:

    -- Take corrective action
    -- Use EPC to return to program

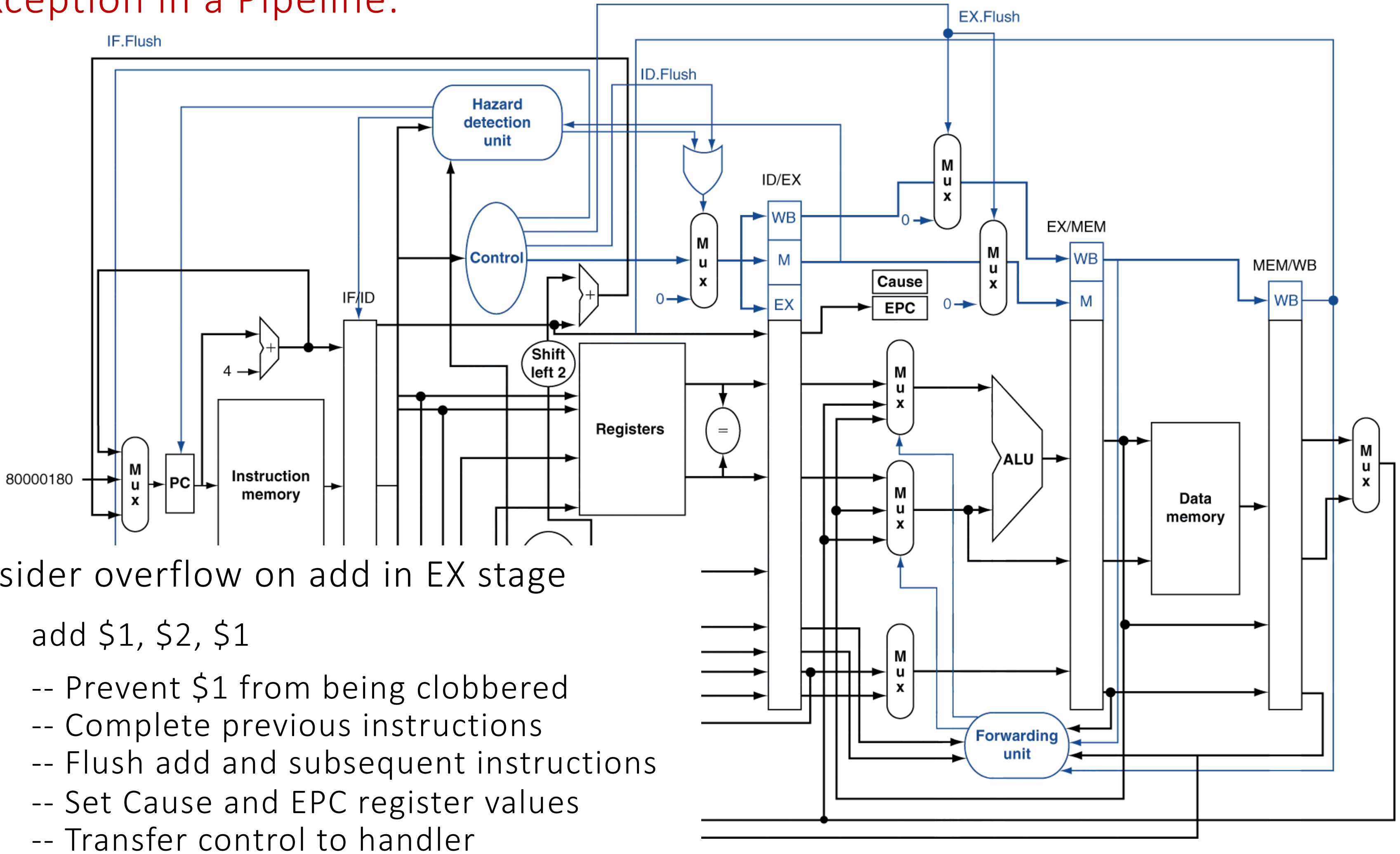
    Otherwise:

    -- Terminate program
    -- Report error using EPC, cause…


User Process
User process executing.
Initiate system call.
Resume execution after the system call has been fully handled by the kernel.
Exception/Trap
User Mode
Kernel Mode
Kernel (OS)
Handle the requested system call.

Exception and Interrupt

- Exception in a Pipeline:



- Consider overflow on add in EX stage

  add \$1, \$2, \$1

  -- Prevent \$1 from being clobbered
  -- Complete previous instructions
  -- Flush add and subsequent instructions
  -- Set Cause and EPC register values
  -- Transfer control to handler

Exception and Interrupt

- Exception properties
  - Restartable exceptions
    - Pipeline can flush the instruction
    - Handler executes, then returns to the instruction
      - Refetched and executed from scratch
  - PC saved in EPC register
    - Identifies causing instruction
    - Actually PC + 4 is saved
      - Handler must adjust

Exception and Interrupt

- Exception Example
  - Exception on add in

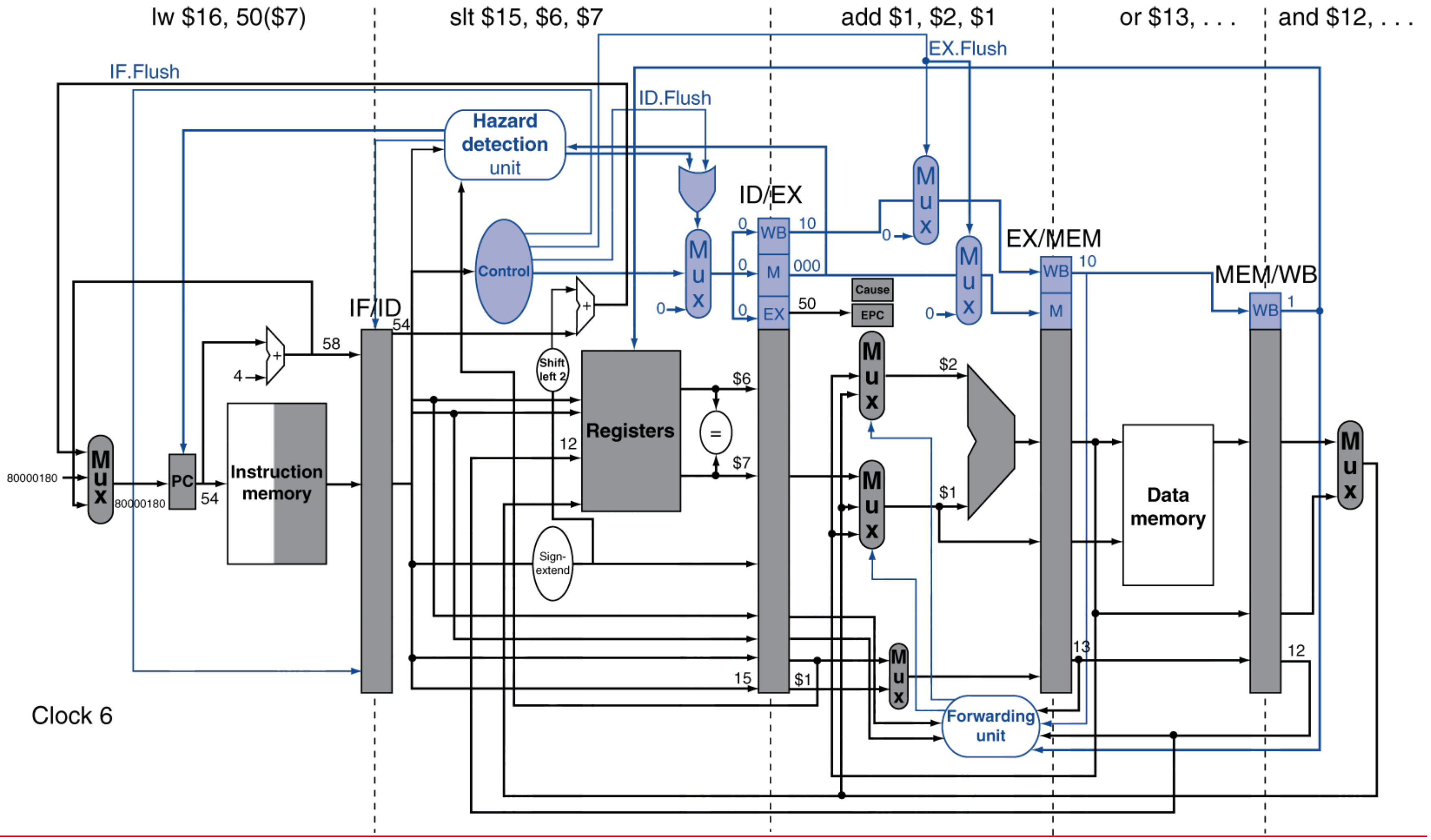
```
40  sub  $11, $2, $4
44  and  $12, $2, $5
48  or   $13, $2, $6
4C  add  $1,  $2, $1
50  slt  $15, $6, $7
54  lw   $16, 50($7)
…
```

  - Handler

```
80000180   sw   $25, 1000($0)
80000184   sw   $26, 1004($0)
```

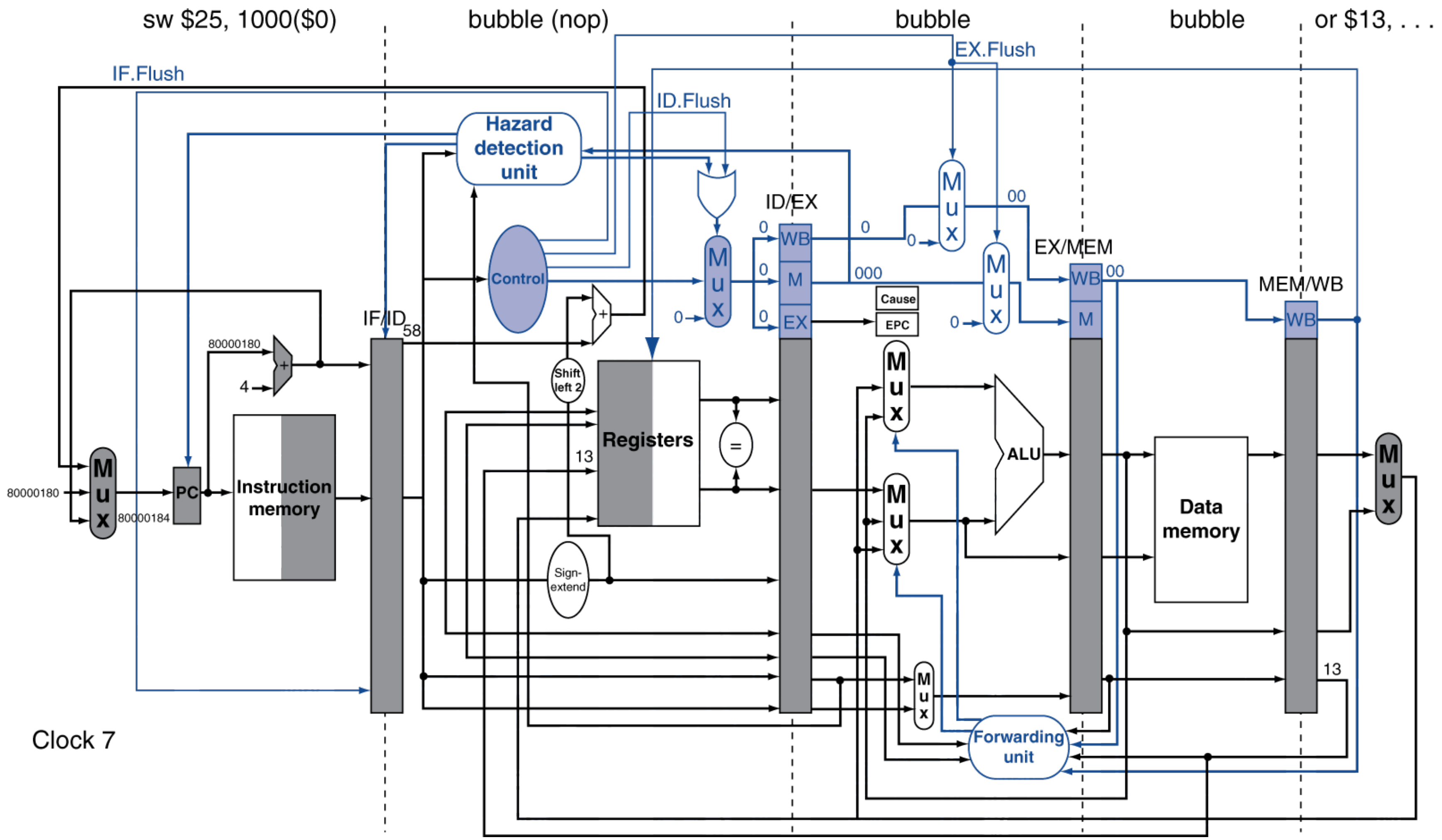
Exception and Interrupt

- Exception Example



Exception and Interrupt

- Exception Example



Exception and Interrupt

Multiple Exception

- Pipelining overlaps multiple instructions
  - Could have multiple exceptions at once
- Simple approach: deal with exception from earliest instruction
  - Flush subsequent instructions
  - "Precise" exceptions
- In complex pipelines
  - Multiple instructions issued per cycle
  - Out-of-order completion
  - Maintaining precise exceptions is difficult!

Stage	**Exception**
IF	memory fault
ID	illegal opcode
EX	arithmetic exception
MEM	memory fault
WB	none

Exception and Interrupt

- Multiple Exception

	CC1	CC2	CC3	CC4	CC5	CC6
lw	IF	ID	EX	MEM*	WB	
add		IF*	ID	EX*	MEM	WB

- Suppose the lw causes a page fault during the MEM stage and the ADD causes a page fault during the IF stage and an arithmetic exception during EX.
- The page fault for the add actually occurs two clock cycles <u>before</u> the page fault for the lw even though the ADD is <u>after</u> the lw in the instruction sequence.

Exception and Interrupt

- Multiple Exception

	CC1	CC2	CC3	CC4	CC5	CC6
lw	IF	ID	EX	MEM*	WB	
add		IF*	ID	EX*	MEM	WB

- Suppose lw causes a page fault during the MEM stage and the ADD causes a page fault during the IF stage and an arithmetic exception during EX.
- Page fault for the add actually occurs two clock cycles before the page fault for the lw even though the ADD is after the lw in the instruction sequence.
- Exceptions can be made precise if exceptions are not handled for any instruction until the instruction gets to the end of the pipeline.
- Since instructions exit pipeline in order, then exceptions will be handled in order.
- This solution requires that information about exceptions caused by an instruction is saved and passed down through the pipeline registers until the end of the pipeline.

Imprecise Exceptions

- Just stop pipeline and save state
  - Including exception cause(s)
- Let the handler work out
  - Which instruction(s) had exceptions
  - Which to complete or flush
    - May require “manual” completion
- Simplifies hardware, but more complex handler software
- Not feasible for complex multiple-issue out-of-order pipelines

What to do when an exception or interrupt occurs

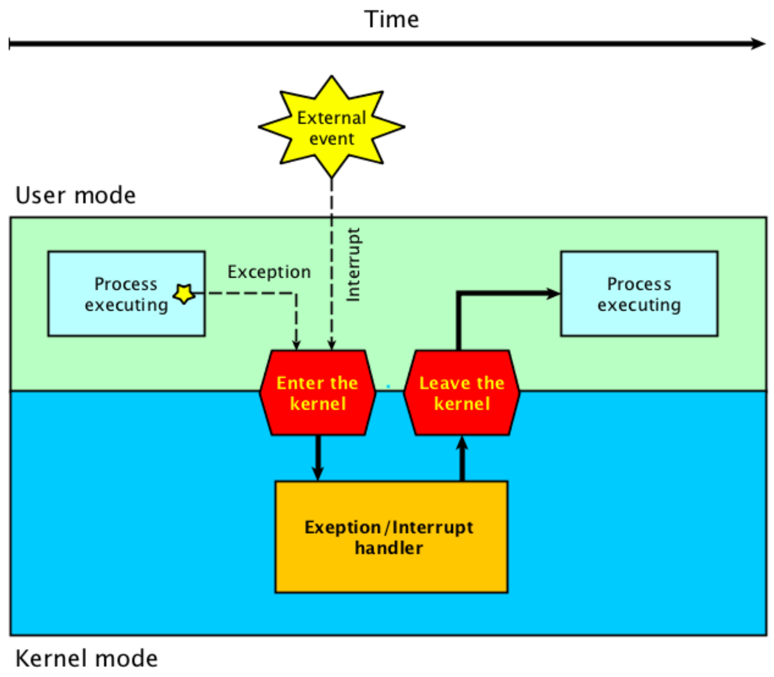
- Just stop pipeline and save state
  - Move the current PC into another register, call the EPC
  - Record the reason for the exception in the Cause register
  - Automatically disable further interrupts or exceptions from occurring, by left-shifting the Status register
  - Change control (jump) to a hardwired exception handler address

MULTIPLE TYPES OF EXCEPTIONS AND INTERRUPTS

- Dealing with multiple types of exceptions and interrupts
  - Record the reason for the exception in the Cause register
  - Automatically disable further interrupts or exceptions from occurring, by left-shifting the Status register
  - Change control (jump) to a hardwired exception handler address

WHAT DID WE LEARN SO FAR

- Interrupt
  - ✓ Actions
  - ✓ Handling
- Exception
  - ✓ Actions
  - ✓ Handling


Time
External event
User mode
Process executing
Exception
Interrupt
Process executing
Enter the kernel
Leave the kernel
Exeption/Interrupt handler
Kernel mode